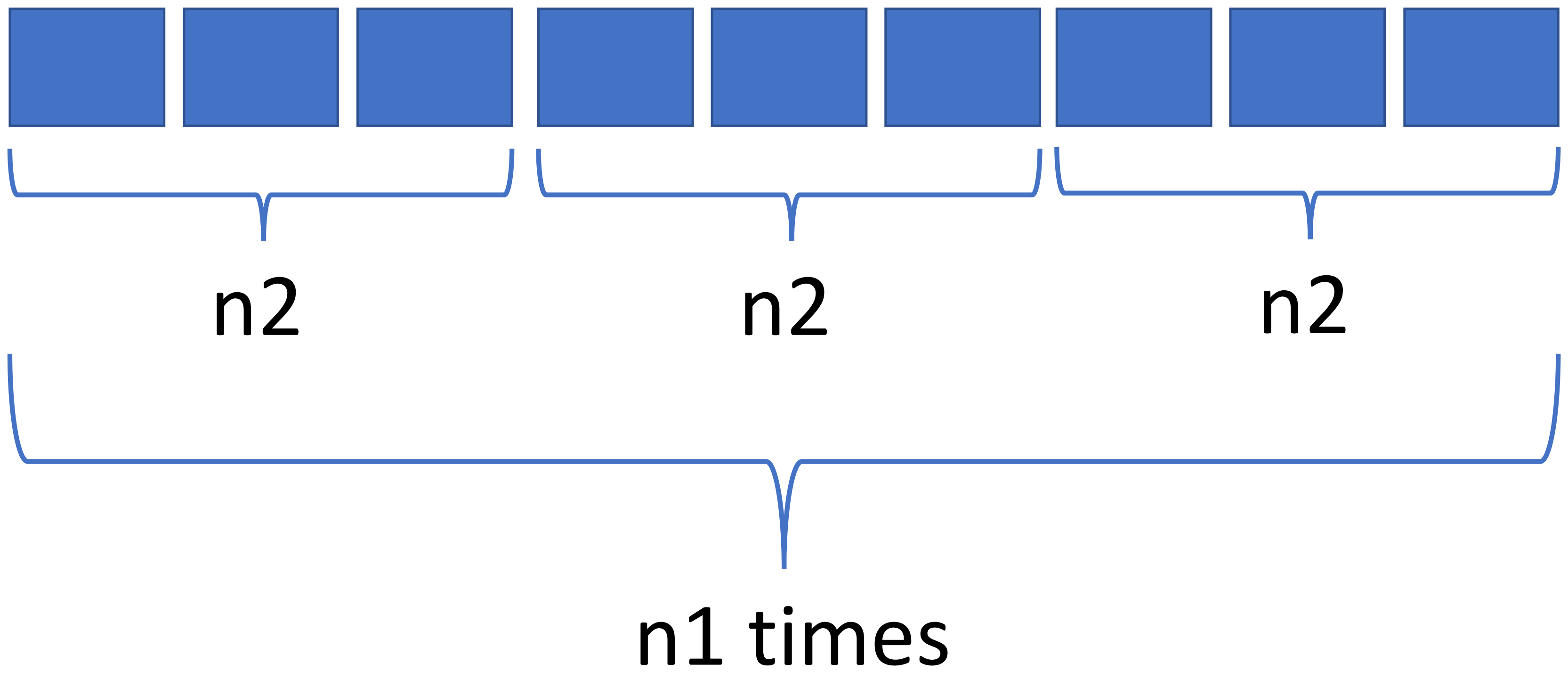
A story about vectorisation and compiler bug report

Arnaud Desitter

C++ On Sea – 16 July 2020


n2
n2
n2
n1 times

```
for (int i1 = 0; i1 < n1; ++i1)
{
  for (int i2 = 0; i2 < n2; ++i2)
  {
    res[i1*n2+i2] = a[i1*n2+i2] - b[i1*n2+i2];
  }
}
```

?

```
for (int index = 0; index < n1*n2; ++index)
{
  res[index] = a[index] - b[index];
}
```

With n1=100,000 and n2=3

	penalty w.r.t. one flattened loop
gcc 9.1 -O3	+50%
gcc 9.1 -O2	+24%
clang 8.0 -O3	+34%
clang 8.0 -O2	+38%

Compilers only vectorise the inner loop.

Let’s try OpenMP

```
for (int i1 = 0; i1 < n1; ++i1)
{  for (int i2 = 0; i2 < n2; ++i2)
   {
     res[i1*n2+i2] = a[i1*n2+i2] - b[i1*n2+i2];
   }
}
```

```
#pragma omp simd collapse(2)
for (int i1 = 0; i1 < n1; ++i1)
{
   for (int i2 = 0; i2 < n2; ++i2)
   {
     res[i1*n1+i2] = a[i1*n1+i2] - b[i1*n1+i2];
   }
}
```

With n1=100,000 and n2=3

	penalty w.r.t. one flattened loop
gcc 9.1 -fopenmp-simd -O3	+100% (1)
clang 7.0 -fopenmp-simd -O3	+1230%
clang 8.0 -fopenmp-simd -O3	**+0%**

(1) https://gcc.gnu.org/bugzilla/show_bug.cgi?id=89371

Let’s try to collapse the loops manually

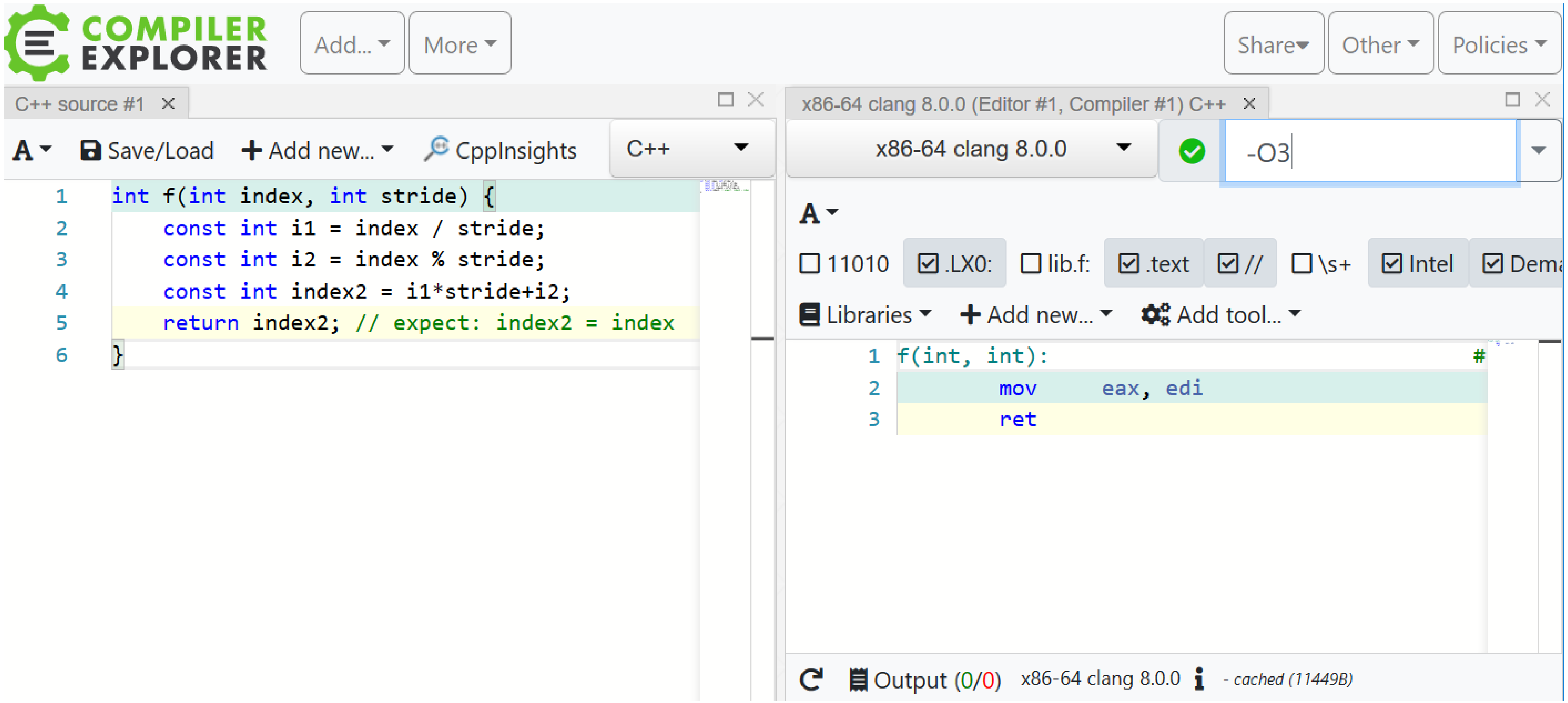
```
for (int i1 = 0; i1 < n1; ++i1)
{ for (int i2 = 0; i2 < n2; ++i2)
  {
    res[i1*n2+i2] = a[i1*n2+i2] - b[i1*n2+i2];
  }
}
```

```
for (int index = 0; index < n1*n2; ++index)
{
  int i1 = index / n2;
  int i2 = index % n2;
  res[i1*n2+i2] = a[i1*n2+i2] - b[i1*n2+i2];
}
```

```
(index / n2) * n2 + index % n2 == index
```

With n1=100,000 and n2=3

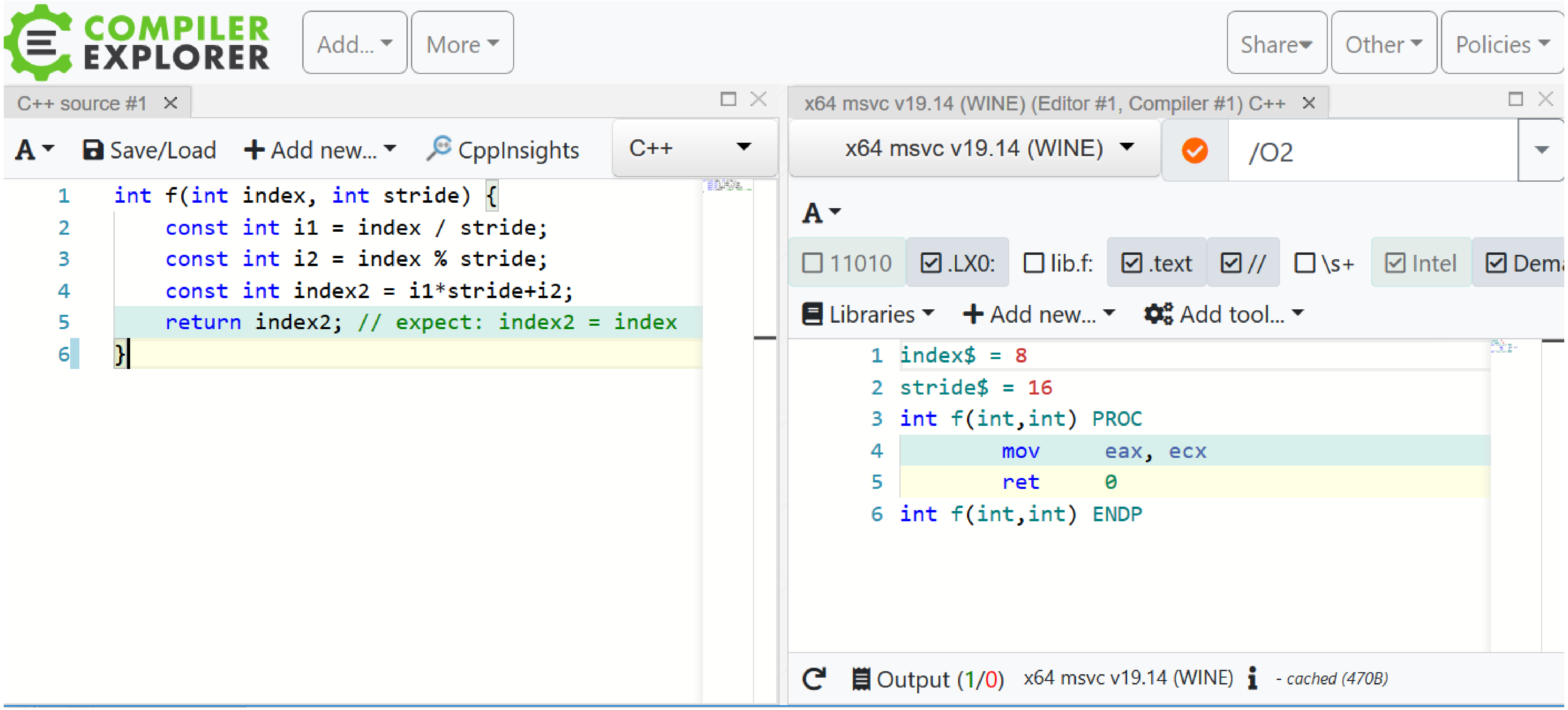
	penalty w.r.t. one flattened loop
gcc 9.1 -O3	+383%
clang 8.0 -O3	**+0%**



```cpp
int f(int index, int stride) {
    const int i1 = index / stride;
    const int i2 = index % stride;
    const int index2 = i1*stride+i2;
    return index2; // expect: index2 = index
}
```

-O3

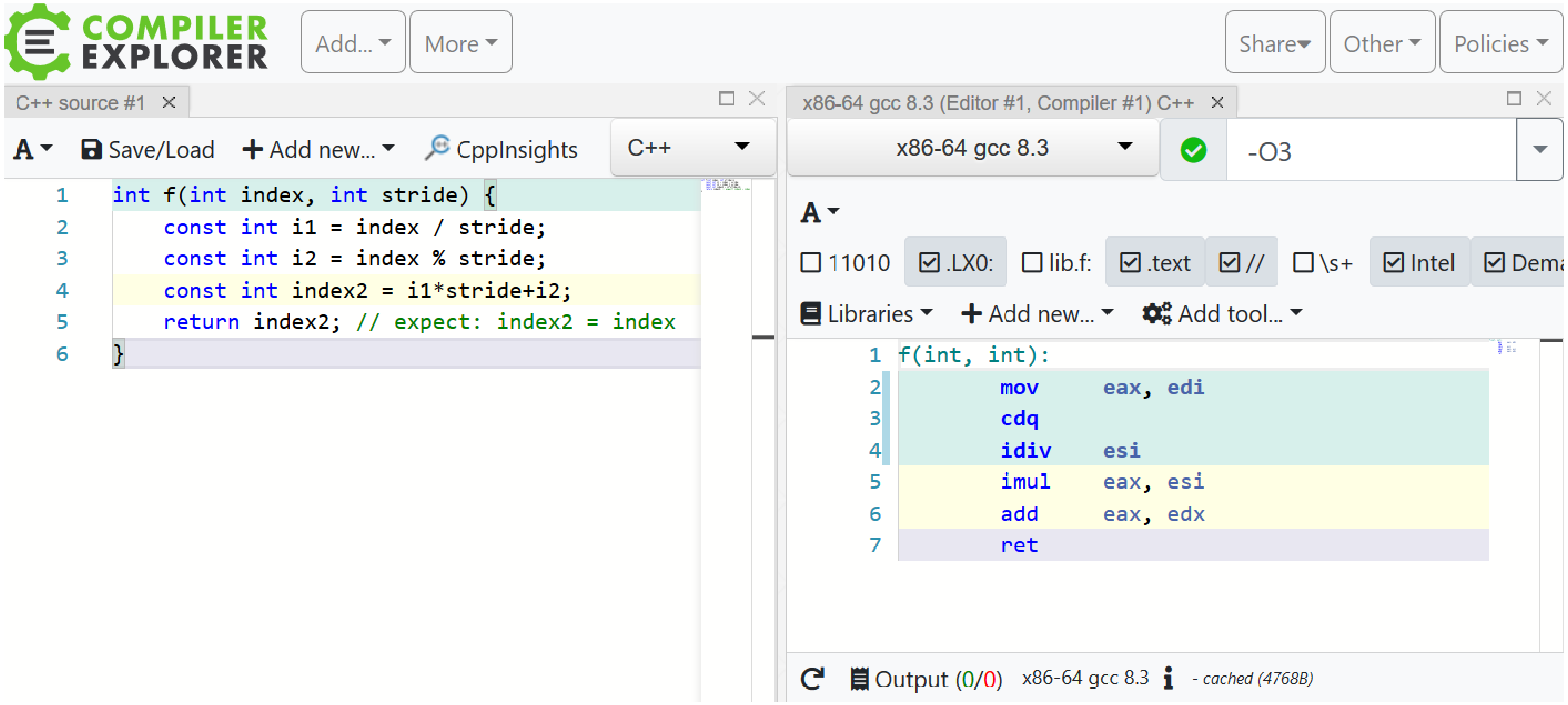
```
f(int, int):                                   #
        mov     eax, edi
        ret
```



Compiler options: /O2

```cpp
int f(int index, int stride) {
    const int i1 = index / stride;
    const int i2 = index % stride;
    const int index2 = i1*stride+i2;
    return index2; // expect: index2 = index
}
```

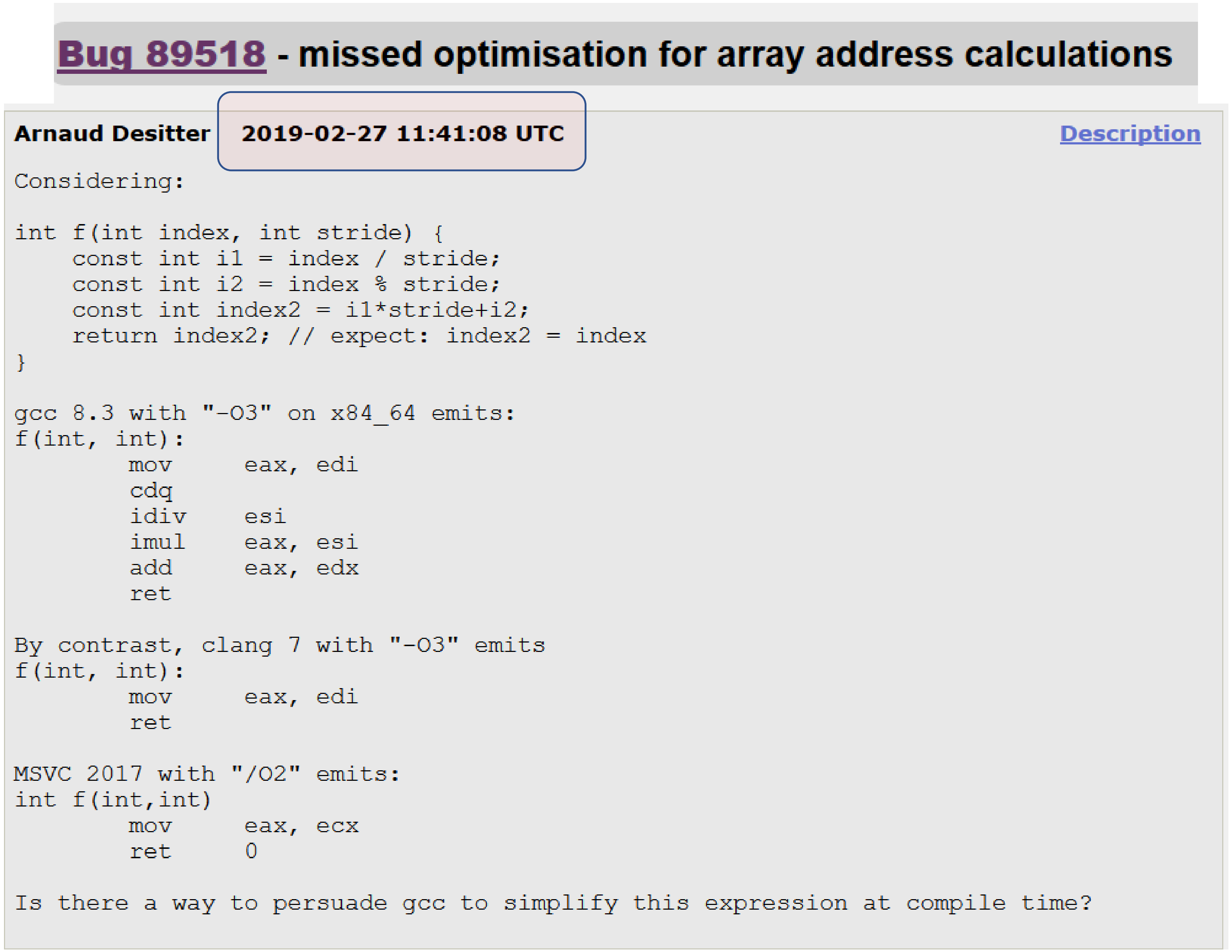
```
index$ = 8
stride$ = 16
int f(int,int) PROC
        mov     eax, ecx
        ret     0
int f(int,int) ENDP
```



-O3

```cpp
int f(int index, int stride) {
    const int i1 = index / stride;
    const int i2 = index % stride;
    const int index2 = i1*stride+i2;
    return index2; // expect: index2 = index
}
```

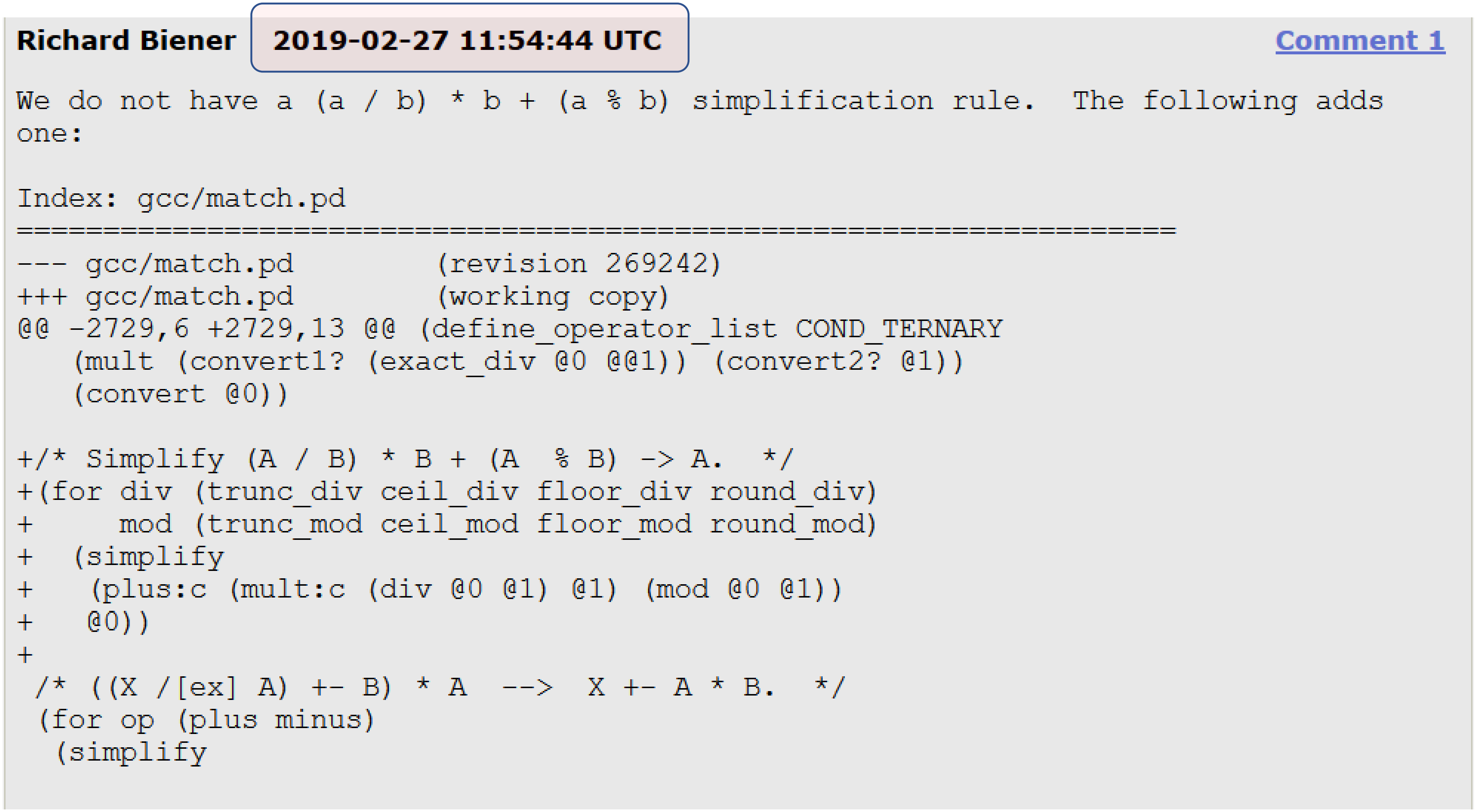
```
f(int, int):
        mov     eax, edi
        cdq
        idiv    esi
        imul    eax, esi
        add     eax, edx
        ret
```



Bug 89518 - missed optimisation for array address calculations

Arnaud Desitter **2019-02-27 11:41:08 UTC** Description

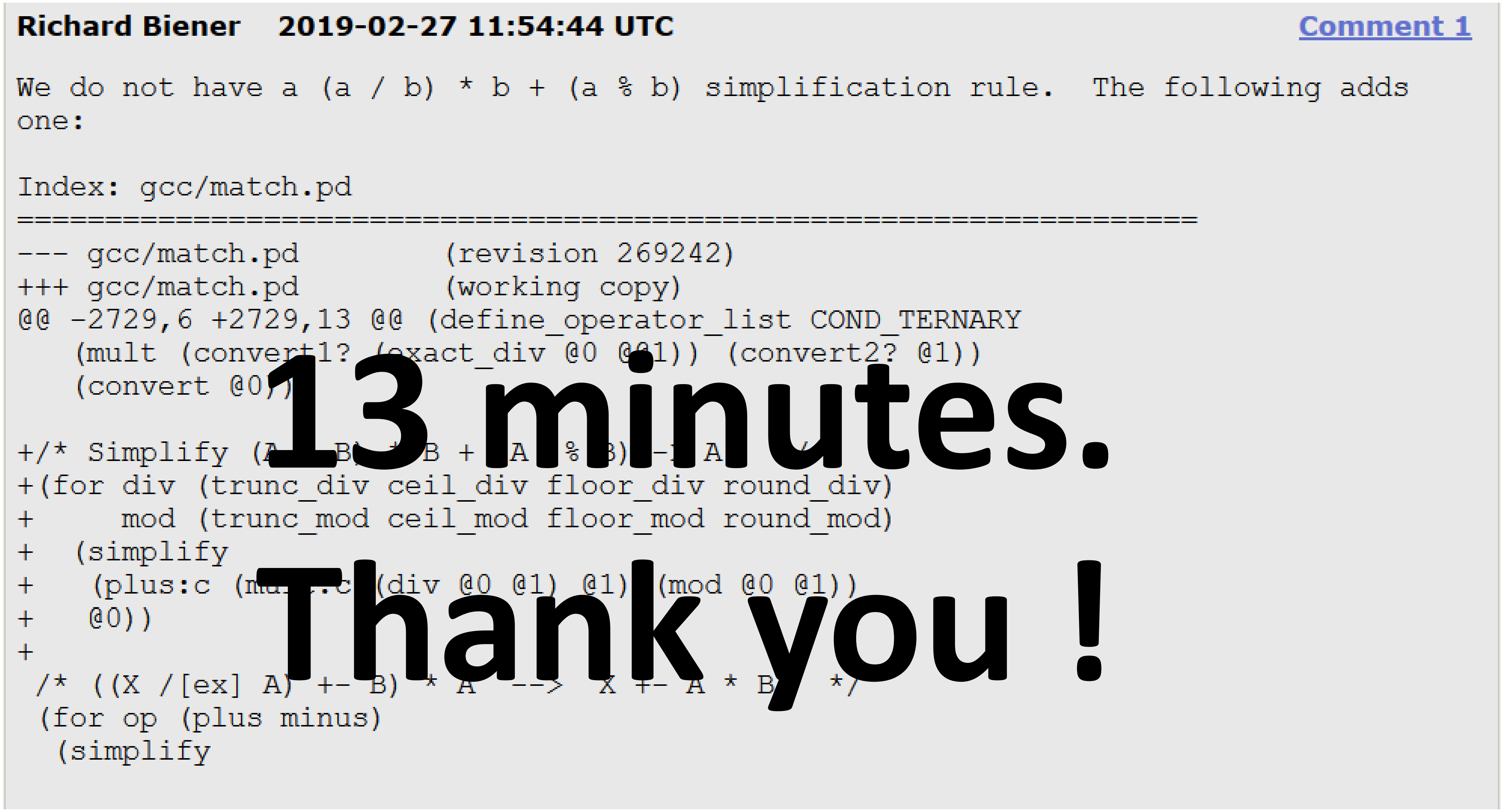
```
Considering:

int f(int index, int stride) {
    const int i1 = index / stride;
    const int i2 = index % stride;
    const int index2 = i1*stride+i2;
    return index2; // expect: index2 = index
}

gcc 8.3 with "-O3" on x84_64 emits:
f(int, int):
        mov     eax, edi
        cdq
        idiv    esi
        imul    eax, esi
        add     eax, edx
        ret

By contrast, clang 7 with "-O3" emits
f(int, int):
        mov     eax, edi
        ret

MSVC 2017 with "/O2" emits:
int f(int,int)
        mov     eax, ecx
        ret     0

Is there a way to persuade gcc to simplify this expression at compile time?
```



Richard Biener 2019-02-27 11:54:44 UTC **Comment 1**

We do not have a (a / b) * b + (a % b) simplification rule. The following adds
one:

```
Index: gcc/match.pd
===================================================================
--- gcc/match.pd        (revision 269242)
+++ gcc/match.pd        (working copy)
@@ -2729,6 +2729,13 @@ (define_operator_list COND_TERNARY
   (mult (convert1? (exact_div @0 @@1)) (convert2? @1))
   (convert @0))
 
+/* Simplify (A / B) * B + (A  % B) -> A.  */
+(for div (trunc_div ceil_div floor_div round_div)
+     mod (trunc_mod ceil_mod floor_mod round_mod)
+  (simplify
+   (plus:c (mult:c (div @0 @1) @1) (mod @0 @1))
+   @0))
+
 /* ((X /[ex] A) +- B) * A  -->  X +- A * B.  */
 (for op (plus minus)
  (simplify
```

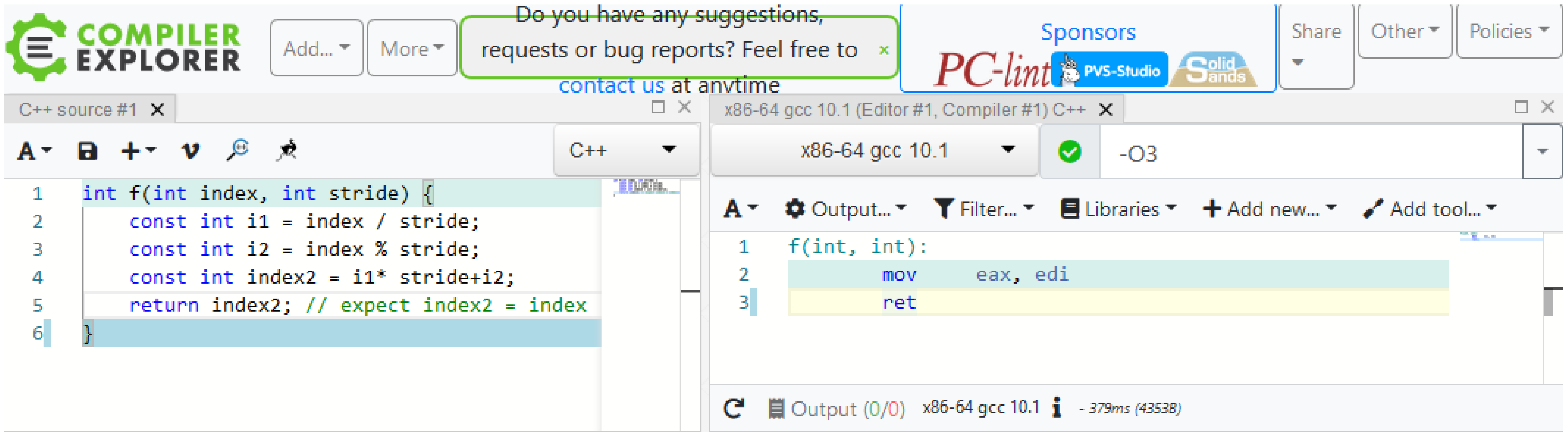

Richard Biener **2019-02-27 11:54:44 UTC** Comment 1

We do not have a (a / b) * b + (a % b) simplification rule. The following adds one:

```
Index: gcc/match.pd
===================================================================
--- gcc/match.pd        (revision 269242)
+++ gcc/match.pd        (working copy)
@@ -2729,6 +2729,13 @@ (define_operator_list COND_TERNARY
   (mult (convert1? (exact_div @0 @@1)) (convert2? @1))
   (convert @0))
 
+/* Simplify (A / B) * B + (A % B) -> A.  */
+(for div (trunc_div ceil_div floor_div round_div)
+     mod (trunc_mod ceil_mod floor_mod round_mod)
+  (simplify
+   (plus:c (mult:c (div @0 @1) @1) (mod @0 @1))
+   @0))
+
 /* ((X /[ex] A) +- B) * A  -->  X +- A * B.  */
 (for op (plus minus)
  (simplify
```

13 minutes.

Thank you !



```cpp
int f(int index, int stride) {
    const int i1 = index / stride;
    const int i2 = index % stride;
    const int index2 = i1* stride+i2;
    return index2; // expect index2 = index
}
```

```asm
f(int, int):
        mov     eax, edi
        ret
```

With n1=100,000 and n2=3

	penalty w.r.t. one flattened loop
gcc 9.1 -O3	+383%
gcc 10.1 -O3	**+0%**
clang 8.0 -O3	**+0%**

Reporting bugs pays off